Formal Verification of an Explicit Counterexample to the Jacobian Conjecture

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

July 21, 2026

Abstract

The Jacobian conjecture asks whether a polynomial self-map of affine space over a field of characteristic zero has a polynomial inverse whenever its Jacobian determinant is a nonzero constant [4, 5]. This entry gives an independent Isabelle/HOL verification of the explicit three-dimensional map announced by Levent Alpöge on July 20, 2026 [1]. The announcement credits Akhil Mathew with prompting the question and the AI system Claude Fable with work leading to the map; stable Lean and independent verification repositories appeared the same day [2, 3].

Isabelle proves that the formal Jacobian entries are the corresponding complex analytic partial derivatives and that the determinant is identically -2. It verifies that three distinct rational points have the same image. Scaling the first output coordinate yields determinant exactly 1 while preserving noninjectivity. The theory defines polynomial invertibility and proves that the map has no polynomial inverse. It also constructs identity-padded polynomial maps, proves their block-Jacobian determinant is unchanged, and thereby obtains counterexamples in every finite dimension at least three. The development makes no claim about the two-dimensional conjecture.

Contents

theory *Jacobian-Counterexample*
imports
Complex-Main
HOL−Analysis.Derivative
HOL−Analysis.Determinants
HOL−Decision-Procs.Commutative-Ring
HOL−Library.Cardinality
begin

1 Formal multivariate polynomials

We use a small expression datatype for multivariate polynomials. This makes polynomiality syntactic and gives a direct definition of formal partial differentiation, independent of analytic differentiability.

datatype $('v, 'a)$ *poly-expr* =
PConst $'a$
| *PVar* $'v$
| *PAdd* $('v, 'a)$ *poly-expr* $('v, 'a)$ *poly-expr*
| *PMul* $('v, 'a)$ *poly-expr* $('v, 'a)$ *poly-expr*
| *PPow* $('v, 'a)$ *poly-expr nat*

instantiation *poly-expr* :: (*type*, *zero*) *zero*
begin

definition *0* = *PConst 0*

instance ⟨*proof*⟩

end

fun *eval-poly-expr* ::
$('v \Rightarrow 'a$::*semiring-1*$) \Rightarrow ('v, 'a)$ *poly-expr* $\Rightarrow 'a$ **where**
eval-poly-expr α (*PConst c*) = *c*
| *eval-poly-expr* α (*PVar v*) = α *v*
| *eval-poly-expr* α (*PAdd p q*) =
eval-poly-expr α *p* + *eval-poly-expr* α *q*
| *eval-poly-expr* α (*PMul p q*) =
eval-poly-expr α *p* ∗ *eval-poly-expr* α *q*
| *eval-poly-expr* α (*PPow p n*) = *eval-poly-expr* α *p* ^ *n*

fun *partial-poly-expr* ::
$'v \Rightarrow ('v, 'a$::*semiring-1*$)$ *poly-expr* $\Rightarrow ('v, 'a)$ *poly-expr* **where**
partial-poly-expr v (*PConst c*) = *PConst 0*
| *partial-poly-expr v* (*PVar w*) = *PConst* (*if v* = *w then 1 else 0*)
| *partial-poly-expr v* (*PAdd p q*) =
PAdd (*partial-poly-expr v p*) (*partial-poly-expr v q*)
| *partial-poly-expr v* (*PMul p q*) =
PAdd (*PMul* (*partial-poly-expr v p*) *q*)

```
      (PMul p (partial-poly-expr v q))
| partial-poly-expr v (PPow p n) =
    PMul (PMul (PConst (of-nat n)) (PPow p (n − 1)))
      (partial-poly-expr v p)

lemma eval-partial-poly-expr-has-field-derivative:
  fixes α :: 'v ⇒ complex
  shows ((λu. eval-poly-expr (α(v := u)) p) has-field-derivative
    eval-poly-expr (α(v := u)) (partial-poly-expr v p)) (at u)
⟨proof⟩

definition eval-poly-map ::
    (('n, 'a::semiring-1) poly-expr)^'n ⇒ 'a^'n ⇒ 'a^'n where
  eval-poly-map P x = (χ i. eval-poly-expr (λj. x $ j) (P $ i))

definition jacobian-matrix ::
    (('n, 'a::comm-ring-1) poly-expr)^'n ⇒ 'a^'n ⇒ 'a^'n^'n where
  jacobian-matrix P x =
    (χ i j. eval-poly-expr (λk. x $ k) (partial-poly-expr j (P $ i)))

lemma jacobian-matrix-entry-has-field-derivative:
  fixes P :: (('n, complex) poly-expr)^'n
  shows ((λu. eval-poly-expr ((λk. x $ k)(j := u)) (P $ i))
    has-field-derivative jacobian-matrix P x $ i $ j) (at (x $ j))
  ⟨proof⟩

definition polynomially-invertible ::
    (('n, 'a::semiring-1) poly-expr)^'n ⇒ bool where
  polynomially-invertible P ⟷
    (∃ Q.
      (∀ x. eval-poly-map Q (eval-poly-map P x) = x) ∧
      (∀ x. eval-poly-map P (eval-poly-map Q x) = x))

lemma polynomially-invertible-imp-inj:
  assumes polynomially-invertible P
  shows inj (eval-poly-map P)
⟨proof⟩

definition keller-map ::
    (('n, 'a::comm-ring-1) poly-expr)^'n ⇒ bool where
  keller-map P ⟷
    (∃ d. d ≠ 0 ∧ (∀ x. det (jacobian-matrix P x) = d))

definition normalized-keller-map ::
    (('n, 'a::comm-ring-1) poly-expr)^'n ⇒ bool where
  normalized-keller-map P ⟷
    (∀ x. det (jacobian-matrix P x) = 1)

definition strong-keller-counterexample ::
```

((′n, ′a::comm-ring-1) poly-expr)^′n ⇒ bool **where**
strong-keller-counterexample P ⟷
keller-map P ∧ ¬ *inj* (*eval-poly-map P*)

definition *keller-counterexample* ::
((′n, ′a::comm-ring-1) poly-expr)^′n ⇒ bool **where**
keller-counterexample P ⟷
keller-map P ∧ ¬ *polynomially-invertible P*

definition *strong-normalized-keller-counterexample* ::
((′n, ′a::comm-ring-1) poly-expr)^′n ⇒ bool **where**
strong-normalized-keller-counterexample P ⟷
normalized-keller-map P ∧ ¬ *inj* (*eval-poly-map P*)

definition *normalized-keller-counterexample* ::
((′n, ′a::comm-ring-1) poly-expr)^′n ⇒ bool **where**
normalized-keller-counterexample P ⟷
normalized-keller-map P ∧ ¬ *polynomially-invertible P*

lemma *strong-keller-counterexample-imp-keller-counterexample*:
strong-keller-counterexample P ⟹ *keller-counterexample P*
⟨*proof*⟩

lemma *strong-normalized-keller-counterexample-imp-counterexample*:
strong-normalized-keller-counterexample P ⟹
normalized-keller-counterexample P
⟨*proof*⟩

2 The announced polynomial map

Alpöge announced the following three-coordinate polynomial map on July 20, 2026 [1]. The announcement credits Akhil Mathew with prompting the question and the AI system Claude Fable with work leading to the example; stable independent verification repositories appeared the same day [2, 3]. The definitions below encode the map syntactically, so the subsequent Jacobian calculation is a theorem about formal polynomial differentiation rather than an imported computer-algebra result.

definition *x-poly* :: (*3*, ′a::comm-ring-1) poly-expr **where**
x-poly = *PVar 1*

definition *y-poly* :: (*3*, ′a::comm-ring-1) poly-expr **where**
y-poly = *PVar 2*

definition *z-poly* :: (*3*, ′a::comm-ring-1) poly-expr **where**
z-poly = *PVar 3*

definition *one-plus-xy-poly* :: (*3*, ′a::comm-ring-1) poly-expr **where**

```
  one-plus-xy-poly = PAdd (PConst 1) (PMul x-poly y-poly)

definition four-plus-three-xy-poly :: (3, 'a::comm-ring-1) poly-expr where
  four-plus-three-xy-poly =
    PAdd (PConst 4) (PMul (PConst 3) (PMul x-poly y-poly))

definition component-a-poly :: (3, 'a::comm-ring-1) poly-expr where
  component-a-poly =
    PAdd (PMul (PPow one-plus-xy-poly 3) z-poly)
      (PMul (PMul (PPow y-poly 2) one-plus-xy-poly)
        four-plus-three-xy-poly)

definition component-b-poly :: (3, 'a::comm-ring-1) poly-expr where
  component-b-poly =
    PAdd y-poly
      (PAdd (PMul (PMul (PConst 3) x-poly)
          (PMul (PPow one-plus-xy-poly 2) z-poly))
        (PMul (PMul (PMul (PConst 3) x-poly) (PPow y-poly 2))
          four-plus-three-xy-poly))

definition component-c-poly :: (3, 'a::comm-ring-1) poly-expr where
  component-c-poly =
    PAdd (PMul (PConst 2) x-poly)
      (PAdd (PMul (PConst (-3)) (PMul (PPow x-poly 2) y-poly))
        (PMul (PConst (-1)) (PMul (PPow x-poly 3) z-poly)))

definition alpoge-map :: ((3, 'a::comm-ring-1) poly-expr)^3 where
  alpoge-map =
    vector [component-a-poly, component-b-poly, component-c-poly]

lemma eval-alpoge-map:
  eval-poly-map alpoge-map (vector [x, y, z]) =
   vector [
    (1 + x * y)^3 * z + y^2 * (1 + x * y) * (4 + 3 * x * y),
    y + 3 * x * (1 + x * y)^2 * z + 3 * x * y^2 * (4 + 3 * x * y),
    2 * x - 3 * x^2 * y - x^3 * z]
  ⟨proof⟩

corollary alpoge-jacobian-entry-has-field-derivative:
  fixes x :: complex^3
  shows ((λu.
      eval-poly-map (alpoge-map :: ((3, complex) poly-expr)^3)
        (χ k. if k = j then u else x $ k) $ i)
    has-field-derivative
      jacobian-matrix (alpoge-map :: ((3, complex) poly-expr)^3) x $ i $ j)
    (at (x $ j))
  ⟨proof⟩

lemma jacobian-alpoge-map:
```

```
  jacobian-matrix alpoge-map (vector [x, y, z]) =
   vector [
     vector [
       3 * y * (1 + x * y)^2 * z +
         y^3 * (4 + 3 * x * y) + 3 * y^3 * (1 + x * y),
       3 * x * (1 + x * y)^2 * z +
         2 * y * (1 + x * y) * (4 + 3 * x * y) +
         x * y^2 * (4 + 3 * x * y) + 3 * x * y^2 * (1 + x * y),
       (1 + x * y)^3],
     vector [
       3 * (1 + x * y)^2 * z + 6 * x * y * (1 + x * y) * z +
         3 * y^2 * (4 + 3 * x * y) + 9 * x * y^3,
       1 + 6 * x^2 * (1 + x * y) * z +
         6 * x * y * (4 + 3 * x * y) + 9 * x^2 * y^2,
       3 * x * (1 + x * y)^2],
     vector [
       2 - 6 * x * y - 3 * x^2 * z,
       -3 * x^2,
       -(x^3)]]
  ⟨proof⟩

lemma alpoge-jacobian-det-coordinates:
  det (jacobian-matrix alpoge-map (vector [x, y, z])) = -2
  ⟨proof⟩

theorem alpoge-jacobian-det:
  det (jacobian-matrix alpoge-map p) = -2
⟨proof⟩
```

3 An explicit three-point fibre

```
definition collision-point-0 :: 'a::field-char-0^3 where
  collision-point-0 = vector [0, 0, -1 / 4]

definition collision-point-pos :: 'a::field-char-0^3 where
  collision-point-pos = vector [1, -3 / 2, 13 / 2]

definition collision-point-neg :: 'a::field-char-0^3 where
  collision-point-neg = vector [-1, 3 / 2, 13 / 2]

definition collision-value :: 'a::field-char-0^3 where
  collision-value = vector [-1 / 4, 0, 0]

lemma collision-points-distinct:
  distinct [collision-point-0, collision-point-pos, collision-point-neg]
  ⟨proof⟩

lemma alpoge-collision:
  eval-poly-map alpoge-map collision-point-0 = collision-value
```

eval-poly-map alpoge-map collision-point-pos = collision-value
eval-poly-map alpoge-map collision-point-neg = collision-value
⟨*proof*⟩

lemma *alpoge-map-not-injective*:
¬ *inj*
(*eval-poly-map* (*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*))
⟨*proof*⟩

theorem *alpoge-map-is-strong-keller-counterexample*:
strong-keller-counterexample
(*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
⟨*proof*⟩

corollary *alpoge-map-is-keller-counterexample*:
keller-counterexample
(*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
⟨*proof*⟩

corollary *alpoge-map-not-polynomially-invertible*:
¬ *polynomially-invertible*
(*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
⟨*proof*⟩

4 A determinant-one normalization

definition *normalized-alpoge-map* ::
((*3*, *'a*::*field-char-0*) *poly-expr*)^*3* **where**
normalized-alpoge-map =
vector [
PMul (*PConst* (−*1* / *2*)) *component-a-poly*,
component-b-poly,
component-c-poly]

lemma *eval-normalized-alpoge-map*:
eval-poly-map normalized-alpoge-map (*vector* [*x*, *y*, *z*]) =
vector [
(−*1* / *2*) *
((*1* + *x* * *y*)^*3* * *z* + *y*^*2* * (*1* + *x* * *y*) * (*4* + *3* * *x* * *y*)),
y + *3* * *x* * (*1* + *x* * *y*)^*2* * *z* + *3* * *x* * *y*^*2* * (*4* + *3* * *x* * *y*),
2 * *x* − *3* * *x*^*2* * *y* − *x*^*3* * *z*]
⟨*proof*⟩

lemma *jacobian-normalized-alpoge-map*:
jacobian-matrix normalized-alpoge-map p =
(χ *i j*.
if i = *1 then*
(−*1* / *2*) * (*jacobian-matrix alpoge-map p* $ *i* $ *j*)
else jacobian-matrix alpoge-map p $ *i* $ *j*)

⟨*proof*⟩

theorem *normalized-alpoge-jacobian-det*:
det (*jacobian-matrix normalized-alpoge-map p*) = *1*
⟨*proof*⟩

lemma *normalized-alpoge-collision*:
eval-poly-map normalized-alpoge-map collision-point-0 =
vector [*1* / *8*, *0*, *0*]
eval-poly-map normalized-alpoge-map collision-point-pos =
vector [*1* / *8*, *0*, *0*]
eval-poly-map normalized-alpoge-map collision-point-neg =
vector [*1* / *8*, *0*, *0*]
⟨*proof*⟩

lemma *normalized-alpoge-map-not-injective*:
¬ *inj*
(*eval-poly-map*
(*normalized-alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*))
⟨*proof*⟩

theorem *normalized-alpoge-map-is-strong-counterexample*:
strong-normalized-keller-counterexample
(*normalized-alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
⟨*proof*⟩

theorem *normalized-alpoge-map-is-counterexample*:
normalized-keller-counterexample
(*normalized-alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
⟨*proof*⟩

corollary *normalized-alpoge-map-not-polynomially-invertible*:
¬ *polynomially-invertible*
(*normalized-alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
⟨*proof*⟩

This is the determinant-one form of the counterexample. Since every coordinate is represented by a polynomial expression, the theorem directly refutes the normalized dimension-three Jacobian conjecture over any field of characteristic zero.

corollary *complex-jacobian-conjecture-counterexample*:
normalized-keller-counterexample
(*normalized-alpoge-map* :: ((*3*, *complex*) *poly-expr*)^*3*)
⟨*proof*⟩

5 Identity padding in higher dimensions

lemma *prod-UNIV-sum*:

fixes $f :: ('n::finite + 'm::finite) \Rightarrow 'a::comm\text{-}monoid\text{-}mult$
shows $(\prod x{\in}UNIV.\ f\ x) =$
$(\prod i{\in}UNIV.\ f\ (Inl\ i)) * (\prod j{\in}UNIV.\ f\ (Inr\ j))$
⟨*proof*⟩

definition *block-identity-matrix* ::
$'a::semiring\text{-}1\hat{}'n\hat{}'n \Rightarrow 'a\hat{}('n + 'm::finite)\hat{}('n + 'm)$ **where**
block-identity-matrix A =
$(\chi\ i\ j.$
case i of
Inl r ⇒ (*case j of Inl c* ⇒ *A* $ *r* $ *c* | *Inr* - ⇒ *0*)
| *Inr r* ⇒ (*case j of Inl* - ⇒ *0* | *Inr c* ⇒ *if r* = *c then 1 else 0*))

lemma *det-block-identity-matrix*:
fixes $A :: 'a::comm\text{-}ring\text{-}1\hat{}'n::finite\hat{}'n$
shows *det* (*block-identity-matrix A* :: $'a\hat{}('n + 'm::finite)\hat{}('n + 'm)$) = *det A*
⟨*proof*⟩

fun *rename-poly-expr* ::
$('v \Rightarrow 'w) \Rightarrow ('v, 'a)\ poly\text{-}expr \Rightarrow ('w, 'a)\ poly\text{-}expr$ **where**
rename-poly-expr f (*PConst c*) = *PConst c*
| *rename-poly-expr f* (*PVar v*) = *PVar* (*f v*)
| *rename-poly-expr f* (*PAdd p q*) =
PAdd (*rename-poly-expr f p*) (*rename-poly-expr f q*)
| *rename-poly-expr f* (*PMul p q*) =
PMul (*rename-poly-expr f p*) (*rename-poly-expr f q*)
| *rename-poly-expr f* (*PPow p n*) = *PPow* (*rename-poly-expr f p*) *n*

lemma *eval-rename-poly-expr*:
eval-poly-expr α (*rename-poly-expr f p*) =
eval-poly-expr ($\alpha \circ f$) *p*
⟨*proof*⟩

lemma *partial-rename-poly-expr-Inl*:
partial-poly-expr (*Inl v*) (*rename-poly-expr Inl p*) =
rename-poly-expr Inl (*partial-poly-expr v p*)
⟨*proof*⟩

lemma *eval-partial-rename-poly-expr-Inr*:
eval-poly-expr α
(*partial-poly-expr* (*Inr w*) (*rename-poly-expr Inl p*)) = *0*
⟨*proof*⟩

definition *join-vector* ::
$'a\hat{}'n \Rightarrow 'a\hat{}'m \Rightarrow 'a\hat{}('n + 'm)$ **where**
join-vector x y =
(χ *i. case i of Inl j* ⇒ *x* $ *j* | *Inr j* ⇒ *y* $ *j*)

lemma *join-vector-Inl* [*simp*]: *join-vector x y* $ *Inl i* = *x* $ *i*

⟨*proof*⟩

lemma *join-vector-Inr* [*simp*]: *join-vector x y* $ *Inr j* = *y* $ *j*
⟨*proof*⟩

lemma *join-vector-comp-Inl* [*simp*]:
($) (*join-vector x y*) ∘ *Inl* = ($) *x*
⟨*proof*⟩

lemma *join-vector-comp-Inr* [*simp*]:
($) (*join-vector x y*) ∘ *Inr* = ($) *y*
⟨*proof*⟩

lemma *join-vector-eq-iff* [*simp*]:
join-vector x y = *join-vector x′ y′* ⟷ *x* = *x′* ∧ *y* = *y′*
⟨*proof*⟩

definition *pad-poly-map* ::
((′*n*, ′*a*::*semiring-1*) *poly-expr*)^′*n* ⇒
((′*n* + ′*m*::*finite*, ′*a*) *poly-expr*)^(′*n* + ′*m*) **where**
pad-poly-map P =
(χ *i*.
case i of
Inl j ⇒ *rename-poly-expr Inl* (*P* $ *j*)
| *Inr j* ⇒ *PVar* (*Inr j*))

lemma *eval-pad-poly-map*:
eval-poly-map (*pad-poly-map P*) (*join-vector x y*) =
join-vector (*eval-poly-map P x*) *y*
⟨*proof*⟩

lemma *jacobian-pad-poly-map*:
jacobian-matrix (*pad-poly-map P*) (*join-vector x y*) =
block-identity-matrix (*jacobian-matrix P x*)
⟨*proof*⟩

corollary *jacobian-det-pad-poly-map*:
det (*jacobian-matrix* (*pad-poly-map P*) (*join-vector x y*)) =
det (*jacobian-matrix P x*)
⟨*proof*⟩

definition *left-subvector* :: ′*a*^(′*n*::*finite* + ′*m*::*finite*) ⇒ ′*a*^′*n* **where**
left-subvector z = (χ *i*. *z* $ *Inl i*)

definition *right-subvector* :: ′*a*^(′*n*::*finite* + ′*m*::*finite*) ⇒ ′*a*^′*m* **where**
right-subvector z = (χ *j*. *z* $ *Inr j*)

lemma *join-vector-subvectors* [*simp*]:
join-vector (*left-subvector z*) (*right-subvector z*) = *z*

⟨*proof*⟩

lemma *pad-poly-map-not-injective*:
fixes P :: (('*n*, '*a*::*semiring-1*) *poly-expr*)^'*n*
assumes ¬ *inj* (*eval-poly-map P*)
shows ¬ *inj*
(*eval-poly-map*
(*pad-poly-map P* ::
(('*n* + '*m*::*finite*, '*a*) *poly-expr*)^('*n* + '*m*)))
⟨*proof*⟩

lemma *strong-normalized-keller-counterexample-pad-poly-map*:
fixes P :: (('*n*, '*a*::*comm-ring-1*) *poly-expr*)^'*n*
assumes *strong-normalized-keller-counterexample P*
shows *strong-normalized-keller-counterexample*
(*pad-poly-map P* ::
(('*n* + '*m*::*finite*, '*a*) *poly-expr*)^('*n* + '*m*))
⟨*proof*⟩

theorem *normalized-alpoge-padding-is-strong-counterexample*:
strong-normalized-keller-counterexample
(*pad-poly-map*
(*normalized-alpoge-map* :: ((*3*, '*a*::*field-char-0*) *poly-expr*)^*3*) ::
((*3* + '*m*::*finite*, '*a*) *poly-expr*)^(*3* + '*m*))
⟨*proof*⟩

theorem *normalized-alpoge-padding-is-counterexample*:
normalized-keller-counterexample
(*pad-poly-map*
(*normalized-alpoge-map* :: ((*3*, '*a*::*field-char-0*) *poly-expr*)^*3*) ::
((*3* + '*m*::*finite*, '*a*) *poly-expr*)^(*3* + '*m*))
⟨*proof*⟩

corollary *complex-jacobian-conjecture-counterexample-in-higher-dimension*:
normalized-keller-counterexample
(*pad-poly-map*
(*normalized-alpoge-map* :: ((*3*, *complex*) *poly-expr*)^*3*) ::
((*3* + '*m*::*finite*, *complex*) *poly-expr*)^(*3* + '*m*))
⟨*proof*⟩

lemma *padded-dimension-card*:
CARD(*3* + '*m*::*finite*) = *3* + *CARD*('*m*)
⟨*proof*⟩

lemma *padded-dimension-greater-than-three*:
3 < *CARD*(*3* + '*m*::*finite*)
⟨*proof*⟩

The extra finite type $'m$ is nonempty, so the padded coordinate type has dimension strictly greater than three. Conversely, every concrete finite

dimension above three can be represented by choosing an extra finite type of the required cardinality. Thus the dimension-three counterexample yields counterexamples in all dimensions $n > 3$. Together with the original theorem this covers every $n \geq 3$. Nothing in this development addresses the two-dimensional conjecture.

end

References

[1] L. Alpöge. An Explicit Counterexample to the Jacobian Conjecture. Post on X, July 2026. https://x.com/__alpoge__/status/2079028340955197566, posted July 20, 2026.

[2] D. Cureton. Levent Alpöge/Fable 5's Counterexample to the Jacobian Conjecture in Lean 4. GitHub repository, July 2026. Immutable commit: https://github.com/deancureton/jacobian/tree/0d4a9212d874226ad81ce5a926becddfa94e6a88.

[3] A. Harrison. Machine-Verified Anatomy of the Alpöge–Mathew–Fable Counterexample to the Jacobian Conjecture. GitHub repository, July 2026. Independent verification and Lean certificate, immutable commit: https://github.com/DrAlexHarrison/jacobian-anatomy/tree/74808fb2e1c1691b0007576ba0508e5e7cdcb1e3.

[4] O.-H. Keller. Ganze Cremona-Transformationen. *Monatshefte für Mathematik und Physik*, 47:299–306, 1939. DOI: https://doi.org/10.1007/BF01695502.

[5] A. van den Essen. *Polynomial Automorphisms and the Jacobian Conjecture*, volume 190 of *Progress in Mathematics*. Birkhäuser, Basel, 2000. DOI: https://doi.org/10.1007/978-3-0348-8440-2.